European Journal of Operational Research 306 (2023) 1126–1139

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

Efficient iterated greedy for the two-dimensional bandwidth minimization problem

Sergio Cavero, Eduardo G. Pardo*, Abraham Duarte

Universidad Rey Juan Carlos, C/Tulipán, s/n, Mstoles, 28933, Madrid, Spain

ARTICLE INFO

Article history:
Received 21 January 2022
Accepted 5 September 2022
Available online 9 September 2022

Keywords:
Heuristics
Graph layout problem
Iterated greedy
Combinatorial optimization
Bandwidth

ABSTRACT

Graph layout problems are a family of combinatorial optimization problems that consist of finding an embedding of the vertices of an input graph into a host graph such that an objective function is optimized. Within this family of problems falls the so-called Two-Dimensional Bandwidth Minimization Problem (2DBMP). The 2DBMP aims to minimize the maximum distance between each pair of adjacent vertices of the input graph when it is embedded into a grid host graph. In this paper, we present an efficient heuristic algorithm based on the Iterated Greedy (IG) framework hybridized with a new local search strategy to tackle the 2DBMP. Particularly, we propose different designs for the main IG procedures (i.e., construction, destruction, and reconstruction) based on the trade-off between intensification and diversification. Additionally, the improvement method incorporates three advanced strategies: an efficient way to evaluate the objective function of neighbor solutions, a tiebreak criterion to deal with "flat landscapes", and a neighborhood reduction technique. Extensive experimentation was carried out to assess the IG performance over state-of-the-art methods, emerging our approach as the most competitive algorithm. Specifically, IG finds the best solutions for all instances considered in considerably less execution time. Statistical tests corroborate the merit of our proposal.

1. Introduction

The Two-Dimensional Bandwidth Minimization Problem (2DBMP) belongs to a family of combinatorial optimization problems denoted as Graph Layout Problems (GLP). GLPs consist of embedding an input graph (also known as a candidate graph) into a host graph by defining a mathematical function that relates (or assigns) the vertices of the input graph to the vertices of the host graph, optimizing a particular objective function. These problems can be classified according to the structure of the host graph. Particularly, the most studied GLPs are those that consider a regular host graph, such as: a path (Pardo, García-Sánchez, Sevaux, & Duarte, 2020; Pardo, Mladenović, Pantrigo, & Duarte, 2013), a cycle (Cavero, Pardo, Laguna, & Duarte, 2021b; Cavero, Pardo, & Duarte, 2022a), or a grid (Lin & Lin, 2010; Rodríguez-García, Sánchez-Oro, Rodriguez-Tello, Monfroy, & Duarte, 2021), among others. Also, GLPs can be classified according to the optimized objective function. Among the most studied ones, we can find the minimization of: the maximum cutwidth (Cavero et al., 2021b; Pardo et al., 2013), the linear arrangement (Petit, 2004; Rodriguez-Tello, Hao, & Torres-Jimenez, 2008a), the maximum Bandwidth (Ren, Hao, Rodriguez-Tello, Li, & He, 2020; Rodriguez-Tello, Hao, & Torres-Jimenez, 2008b), or the sum of the Bandwidth (Cavero, Pardo, Duarte, & Rodriguez-Tello, 2022b; Rodriguez-Tello, Narvaez-Teran, & Lardeux, 2019), among others. We refer the interested reader to surveys (Díaz, Petit, & Serna, 2002) and (Pardo, Martí, & Duarte, 2016) for further references about GLPs.

In this paper, we deal with the 2DBMP, which consists of minimizing the Bandwidth of the input graph when embedding it into a grid host graph. The 2DBMP has a large interest for the scientific community from either a practical and theoretical perspective. On the one hand, real-world applications have been devised in the literature related to the problem, as compiled in Section 1.2. On the other hand, from a theoretical perspective, there is a wide family of optimization problems related to the embedding of graphs in regular structures (Ren, Hao, & Rodriguez-Tello, 2019; Rodriguez-Tello et al., 2008b). In this sense, the algorithmic strategies proposed for the 2DBMP have interest, not only for this problem, but also for other related variants. In the following sections, we formally define the 2DBMP (Section 1.1), as well as we review the state of the art of the problem (Section 1.2).

* Corresponding author.
E-mail addresses: sergio.cavero@urjc.es (S. Cavero), eduardo.pardo@urjc.es (E.G. Pardo), abraham.duarte@urjc.es (A. Duarte).

https://doi.org/10.1016/j.ejor.2022.09.004

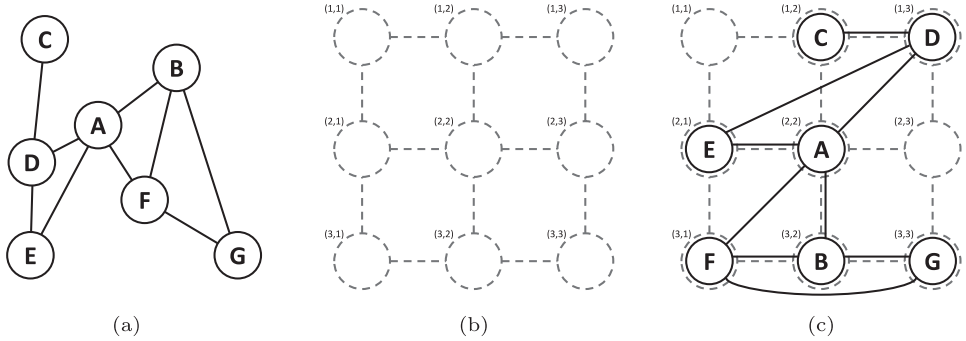

Fig. 1. (a) An input graph, G. (b) A host graph, H. (c) An example of an embedding.

1.1. Problem statement

Before formalizing this problem, we introduce a basic notation. Let $G(V_G, E_G)$ be a connected, unweighted and undirected input graph where the set of vertices is denoted as V_G (with $|V_G| = n$) and its edge set as E_G.

Similarly, let $H = (V_H, E_H)$ be a two-dimensional grid host graph where the set of vertices is denoted as V_H (with $|V_H| = \lceil\sqrt{n}\rceil \cdot \lceil\sqrt{n}\rceil$) and its edge set as E_H. Grid graphs are a common type of lattice graph, whose drawing in a Euclidean space $\mathbb{R}^2$ forms a regular tiling. These graphs satisfy the requirement that each vertex can be represented as a 2-tuple (i, j) that corresponds to a point in the plane, where $1 \leq i, j \leq \lceil\sqrt{n}\rceil$. Two vertices are connected with an edge as long as the corresponding points are at distance 1. Therefore, this particular host graph is a unit-distance, median, and bipartite graph.

In Fig. 1a, we depict an example of an input graph, G, with 7 vertices ($V_G = \{A, B, C, D, E, F, G\}$) and 9 edges ($E_G = \{(A, B), (A, D), (A, E), (A, F), (B, F), (B, G), (C, D), (D, E), (F, G)\}$). Similarly, in Fig. 1b, we show the host graph, H, needed for the embedding of the graph depicted in Fig. 1a (i.e., $|V_H| = 9 = \lceil\sqrt{7}\rceil \cdot \lceil\sqrt{7}\rceil$). In this case, vertices are denoted by their (i, j) coordinates (i.e., $V_H = \{(1, 1), (1, 2), \ldots, (3, 3)\}$), and there is an edge between them if they are at distance 1.

As it was aforementioned, an embedding consists of defining a mathematical function that assigns each vertex of the input graph to a vertex of the host graph. In mathematical terms, let φ be an injective function such that:

$$\varphi : V_G \to V_H,\ \forall\, u \in V_G\ \exists!\ v \in V_H \mid \varphi(u) = v. \tag{1}$$

Since each vertex $v \in V_H$ is defined by a 2-tuple (i, j), for the sake of convenience, we reformulate φ as follows:

$$\varphi : V_G \to V_H,\ \forall\, u \in V_G\ \exists!\ (i, j) \in V_H \mid \varphi(u) = (i, j). \tag{2}$$

with $i, j \in [1, \ldots, \lceil\sqrt{n}\rceil]$.

Then, given an embedding φ of an input graph G, the objective function of the 2DBMP, denoted as BW, is defined as follows:

$$\mathrm{BW}(G, \varphi) = \max_{(u,v)\in E_G} \{d(\varphi(u), \varphi(v))\}, \tag{3}$$

where d is a function that measures the distance between two adjacent vertices. In the related literature, d is computed with the L_1-norm (Lin & Lin, 2010; Rodríguez-García et al., 2021), which is also known as Taxicab norm distance or Manhattan distance (Craw, 2010; Lin & Lin, 2010). That is, the distance between two points $(i, j), (i', j') \in V_H$ is

$$d((i, j), (i', j')) = |i - i'| + |j - j'|. \tag{4}$$

Finally, the Two-Dimensional Bandwidth Minimization Problem (2DBMP) for a graph G consists of finding an embedding $\varphi^\star$ that minimizes Eq. (3). More formally,

$$\varphi^\star \leftarrow \text{2DBMP}(G) = \min_{\varphi\in\Phi} \{\mathrm{BW}(G, \varphi)\}, \tag{5}$$

where Φ represents the set of all possible embeddings of the problem.

In Fig. 1c, we show a possible embedding φ of G (Fig. 1a) in H (Fig. 1b). As can be observed, all vertices of V_G have been assigned to a vertex of V_H through the definition of φ. For example, $\varphi(A) = (2, 2)$ indicates that vertex $A \in V_G$ is assigned to vertex $(2, 2) \in V_H$. Similarly, $\varphi(B) = (2, 3)$ indicates that vertex $B \in V_G$ is assigned to vertex $(3, 2) \in V_H$, and so on.

In order to evaluate the objective function of the example described in Fig. 1, it is required to calculate the distance between each pair of adjacent vertices in V_G (i.e., for each edge of E_G) by using Eq. (4). For instance, considering vertices A and B, with $\varphi(A) = (2, 2)$ and $\varphi(B) = (2, 3)$, the associated distance $|2 - 2| + |2 - 3| = 1$. Similarly, the distance between vertices A and D is 2, since $\varphi(D) = (1, 3)$, and $|2 - 1| + |2 - 3| = 2$. This calculation is performed over the rest of the edges of G. Then, the value of the objective function is the maximum across all distances, which is 3 in this example. Therefore, in mathematical terms, the evaluation of $\mathrm{BW}(G, \varphi)$ in Fig. 1c is computed as follows:

$$\begin{aligned}\mathrm{BW}(G, \varphi) = \max\{&d(\varphi(A), \varphi(B)), d(\varphi(A), \varphi(D)), d(\varphi(A), \varphi(E)),\\ &d(\varphi(A), \varphi(F)), d(\varphi(B), \varphi(F)), d(\varphi(B), \varphi(G)),\\ &d(\varphi(C), \varphi(D)), d(\varphi(D), \varphi(E)), d(\varphi(F), \varphi(G))\}\\ = \max\{&1, 2, 1, 2, 1, 1, 1, 3, 2\} = 3.\end{aligned} \tag{6}$$

1.2. Literature review

The 2DBMP has been widely used to formulate a variety of real-world applications. In particular, it has a direct application in the design of telecommunication architectures, where grid network topologies have gained relevance. These networks are commonly used due to their simple structure, becoming a design that is easy to build and extend (Bezrukov, Chavez, Harper, Röttger, & Schroeder, 1998). The 2DBMP has also been used for very large-scale integration (VLSI) circuit modeling (Bhatt & Thomson Leighton, 1984; Chung, 1988). Indeed, the L_1-norm distance was originally derived from circuit design models where connectors are placed in horizontal or vertical directions, although the paths in the VLSI design cannot overlap each other (Bhatt & Thomson Leighton, 1984; Lin & Lin, 2010). Other practical applications include job scheduling for parallel processing computers, solving systems of equations, or performing matrix decomposition, among others (Lai & Williams, 1999; Rodríguez-García et al., 2021).

The 2DBMP is closely related to other optimization problems belonging to the Graph Layout family, such as the Bandwidth Minimization Problem (BMP) and the Cyclic Bandwidth Problem (CBP). Differences among these three problems reside on the host graph. Specifically, in the BMP, the host graph is a path, while in the CBP it is a cycle, and in the 2DBMP it is a grid.

BMP and CBP have been widely studied by the scientific community. The former was proved to be $\mathcal{NP}$-complete for general graphs in Papadimitriou (1976) and it has been approached from both, exact (Del Corso & Manzini, 1999; Gurari & Sudborough, 1984) and heuristic perspectives (Mladenovic, Urosevic, Pérez-Brito, & García-González, 2010; Rodriguez-Tello et al., 2008b). Similarly, the CBP is also $\mathcal{NP}$-complete for general graphs as proven in Lin (1994). It has been mainly approached by considering special graphs (such as grids, trees, or planar graphs, among others) and determining either lower or upper bounds (Hromkovič, Müller, Sỳkora, & Vrt'o, 1992; Jinjiang & Sanming, 1995). Recently, in Ren et al. (2019, 2020) two advanced metaheuristics have been proposed for the CBP.

In this paper, we focus on the 2DBMP, originally proposed in Chung (1988), that belongs to the $\mathcal{NP}$-complete class (Bhatt & Thomson Leighton, 1984; Lin & Lin, 2010). This optimization problem has been studied from different points of view. In particular, lower bounds for regular-structured graphs were introduced in Lin & Lin (2010, 2011). More recently, three Constraint Satisfaction Programming (CSP) models (Tsang, 2014) were described in Rodríguez-García et al. (2021). The best model, denoted as M3, is able to solve small and regular-structured graphs, providing a lower bound (not necessarily tight) for medium and large instances. The best previous metaheuristic procedure identified in the related literature was introduced in Rodríguez-García et al. (2021). Specifically, the authors described an algorithm based on the combination of the Greedy Randomized Adaptive Search Procedure (GRASP) (Feo & Resende, 1995) and the Basic Variable Neighborhood Search (BVNS) (Hansen & Mladenović, 2006) methodologies. This procedure follows a multi-start strategy, where many initial points are generated with GRASP and then improved with BVNS. The constructive procedure uses a greedy criterion based on the quality of the objective function, while the BVNS, based on the idea of systematic changes of neighborhood the structure within the search, uses two neighborhoods and a random perturbation to escape from local optima. This procedure is currently considered the state of the art for the 2DBMP.

1.3. Our contributions

The main contribution of this work is the proposal of an efficient procedure based on the Iterated Greedy framework. The proposed algorithm includes novel construction/destruction/reconstruction strategies, as well as advanced improvement methods. These techniques are designed from a general perspective, i.e., they are not only valid for the 2DBMP, but also for any other related optimization problem. Specifically, the proposed construction, destruction, and reconstruction strategies vary from totally random to totally greedy approaches. In addition, a straightforward design of a local search for the 2DBMP is enriched by: a tiebreak criterion to distinguish between same-quality solutions; fast evaluation of the objective function; and neighborhood reduction techniques.

The proposed method is configured by a set of preliminary experiments, which allow us tuning the search parameters as well as to evaluate the influence of the proposed mechanisms. Finally, the best identified variant is compared with state-of-the-art algorithms through competitive tests. The merit of the results obtained is supported by statistical tests.

The rest of the paper is organized as follows: Section 2 describes our algorithmic proposal. Section 3 introduces several advanced search strategies. Section 4 presents and analyses the results of the computational experiments carried out. Finally, the conclusions are drawn in Section 5.

2. Algorithmic proposal: Iterated greedy

In this paper, we propose a procedure based on the Iterated Greedy (IG) metaheuristic (Ruiz & Stützle, 2007; Stützle & Ruiz, 2018). IG is a search method where solutions are gradually improved through the repeated application of two main phases: a partial destruction of a solution followed by a reconstruction to reach a new feasible solution. These two phases are usually repeated for a fixed number of iterations, for a maximum number of iterations without finding an improvement, or even for a combination of the two previous criteria.

IG can be easily hybridized with other strategies, such as local search procedures or other metaheuristics. In this case, the destruction and reconstruction phases of IG can be understood as a way to perturb the incumbent solution, similarly to other well-known metaheuristics such as Iterated Local Search (ILS) (Lourenço, Martin, & Stützle, 2003) or Variable neighborhood Search (VNS) (Hansen, Mladenović, Todosijević, & Hanafi, 2017). However, in the IG methodology, an important part of the process, the reconstruction phase, uses greedy decisions rather than stochastic ones. See Stützle & Ruiz (2018) for further details.

In this paper, we propose the hybridization of IG with an efficient local search procedure. The pseudocode of the proposed method is presented in Algorithm 1. The procedure receives as

Algorithm 1 General procedure based on IG algorithm.

```
1:  Procedure IteratedGreedy (G, maxIter, maxNotImprIter)
2:  iter = 0, notImprIter = 0
3:  φ = GreedyConstructive(G)
4:  φ ← LocalSearch(G, φ)
5:  while iter < maxIter do
6:      iter = iter + 1, notImprIter = notImprIter + 1
7:      φ' ← Destruction(notImprIter, φ)
8:      φ'' ← Reconstruction(G, φ')
9:      φ''' ← LocalSearch(G, φ'')
10:     if BW(G, φ''') < BW(G, φ) then
11:         φ ← φ'''
12:         notImprIter = 0
13:     end if
14:     if notImprIter > maxNotImprIter then
15:         break
16:     end if
17: end while
18: return φ
```

parameters: the input graph, G; the maximum number of iterations, *maxIter*; and the number of iterations without improvement *maxNotImprIter*. The algorithm starts by generating an initial solution with the greedy constructive procedure (line 3) that will be introduced in Section 2.1. Then, after obtaining an improved solution through the local search procedure (line 4), described in Section 2.2, the procedure enters a loop (lines 5 to 17). In each iteration, some elements are removed from the current solution using the destruction method (line 7). Next, the solution is greedily reconstructed (line 8) and improved again by the local search procedure (line 9). Both destruction and reconstruction methods, are described in Section 2.3. In each iteration, IG determines (steps 10 to 13) whether the perturbed and improved solution (φ''') is better than the incumbent one (φ). If so, φ and *notImprIter* are updated accordingly. These three last steps (destruction, reconstruction, and local search) are repeated until a maximum number of iterations is reached, unless the procedure is not able to improve the current best solution for a number of iterations (line 14). Once the termination condition is met, IG returns the best solution found (step 18).

2.1. Greedy constructive procedure

Constructing a solution for the 2DBMP from a greedy perspective consists of performing the best possible assignation of the vertices of the input graph to the vertices of the host graph (i.e., defining φ). To do this, we need to answer three questions: #1 which vertices (from either the host or the input graphs) are more suitable to start with; #2 given a partial solution, which vertex (u) of the input graph should be assigned next; and #3 given a partial solution and u, which vertex v of the host graph should be assigned to u. To answer each of these questions, we propose different strategies, which are described below.

We start by selecting a random vertex from the input graph. Then, to perform its assignation, we need to select a vertex from the host graph. In this case, we study three alternatives: a random vertex; a vertex placed in a corner of the grid (i.e., $(1,1), (1, \lceil\sqrt{n}\rceil), (\lceil\sqrt{n}\rceil, 1), (\lceil\sqrt{n}\rceil, \lceil\sqrt{n}\rceil)$; or a vertex placed in the center of the grid (i.e., $(\lceil\frac{\lceil\sqrt{n}\rceil}{2}\rceil, \lceil\frac{\lceil\sqrt{n}\rceil}{2}\rceil)$ if $\lceil\sqrt{n}\rceil$ is odd; or $(\lceil\frac{\lceil\sqrt{n}\rceil}{2}\rceil, \lceil\frac{\lceil\sqrt{n}\rceil}{2}\rceil)$, $(\lceil\frac{\lceil\sqrt{n}\rceil}{2}\rceil + 1, \lceil\frac{\lceil\sqrt{n}\rceil}{2}\rceil)$, $(\lceil\frac{\lceil\sqrt{n}\rceil}{2}\rceil, \lceil\frac{\lceil\sqrt{n}\rceil}{2}\rceil + 1)$, $(\lceil\frac{\lceil\sqrt{n}\rceil}{2}\rceil + 1, \lceil\frac{\lceil\sqrt{n}\rceil}{2}\rceil + 1)$ if n is even).

Given the first assignation, we already have a partial solution. Then, we propose a greedy function, inspired by McAllister et al. (1999) and denoted as g_1, to determine which vertices of the input graph should be assigned in the following steps to a partial solution φ. In other words, g_1 is used to evaluate the "urgency" of any unassigned vertex from the input graph to be assigned next.

Let us introduce some notation before defining g_1. We define the subset $U_G \subset V_G$ as the set of vertices of the input graph that have not been assigned yet in φ. Then, given a vertex $u \in U_G$, we define $A(u)$ as the set of adjacent vertices to u already assigned. More formally, $A(u) = \{v \in V_G : v \notin U_G \wedge (u, v) \in E_G\}$. Similarly, we define $R(u)$ as the set of vertices adjacent to u that remain unassigned. In mathematical terms, $R(u) = \{v \in V_G : v \in U_G \wedge (u, v) \in E_G\}$. It is worth mentioning that $A(u) \cup R(u)$ is the set of adjacent vertices to u. Then, g_1 is defined in Eq. (7) as follows:

$$g_1(u, \varphi) = w_1 \cdot |A(u)| - w_2 \cdot |R(u)|, \tag{7}$$

where w_1 and w_2 are parameters that should be tuned experimentally and satisfy $0 \leq w_1, w_2 \leq 1$ and $w_1 + w_2 = 1$. These two parameters balance the relevance of having a large number of adjacent vertices assigned ($w_1 > w_2$) or a reduced number of adjacent vertices unassigned ($w_1 < w_2$). Notice that if $w_1 = w_2$, then the strategy is equivalent to the original proposal introduced in McAllister et al. (1999). Then, all unassigned vertices from the input graph are evaluated, and the vertex with the largest g_1 value is chosen to be assigned next (with ties broken at random).

Once the vertex of the input graph has been chosen, the second proposed question is answered. Then, an available vertex from the host graph must be selected to embed the selected vertex of the input graph. In this case, we propose two approaches: one based on graphical patterns and the other based on a greedy function. The first approach determines the order in which host vertices are selected on the basis of a graphical pattern. Moreover, in this work we study three patterns: Sequential, Diagonal, and Zigzag, which are illustrated in Fig. 2(a)–(c), respectively. In each of the figures, the sequence is indicated by green arrows and numbers inside each of the host vertices, being the number 1 the vertex selected in the first iteration and number 9 the vertex selected in the last iteration.

The second approach to select a vertex of the host graph is based on a new greedy function, denoted as g_2, that evaluates the variation of the objective function when assigning a vertex of the input graph. As it is well documented in the related literature (Cavero et al., 2021b), some successful strategies in graph layout problems are related to the closeness/remoteness of adjacent vertices. In this case, g_2 is used to place every candidate vertex as close as possible to its adjacent vertices. Let $C_H \subseteq V_H$ be the set of vertices which distance (see Eq. (4)) to the already assigned host vertices is 1. More formally, $C_H(\varphi) = \{(i, j) \in V_H : \forall\, v \notin U_G, d(\varphi(v), (i, j)) = 1\}$. Then, given a vertex $u \in U_G$, a vertex $(i, j) \in C_H$, and a partial solution φ, g_2 is formally defined as:

$$g_2(\varphi, u, (i, j)) = \max_{v \in A(u)} \{d((i, j), \varphi(v))\}. \tag{8}$$

Thus, all unassigned vertices from the host graph are evaluated and the vertex $(i, j) \in C_H(\varphi)$ that minimizes g_2 is selected to host the input graph vertex u, i.e., $\varphi(u) = (i, j)$. The rationale behind this strategy is that the returned value from g_2 corresponds to the contribution of the evaluated vertex to the objective function. Therefore, the best host vertex for an input vertex is the one that minimizes Eq. (8).

In Algorithm 2 we show the pseudocode of a general greedy constructive procedure which can use any of the greedy strategies described in this section. Particularly, it receives an input graph $G = (V_G, E_G)$ and a host graph $H = (V_H, E_H)$ as input parameters. The method starts by selecting a vertex u of the input graph at random (line 2). According to the aforementioned Question #1, in line 3, the procedure identifies the vertex of the host graph to embed u. Then, the first assignation is performed (line 4). The set of unassigned vertices U_G for the partial solution φ is constructed in line 5. While there are still elements in U_G, the greedy constructive procedure selects a vertex according to g_1 (see Question #2) in line 8. Next, this selected vertex is assigned to one vertex in the host graph (see Question #3), either based on patterns (Fig. 2) or based on g_2. Finally, the partial solution and the set of unassigned vertices are updated in lines 10 and 11, respectively. Lines 7 to 12 are repeated until generating a complete feasible solution, which is returned at the end of the procedure (line 13). To complement the pseudocode, we graphically illustrate the steps of the constructive procedure in the flowchart depicted in Fig. 3. This flowchart follows the activity diagram standard described in the Unified Modeling Language (Booch, 2005). Therefore, each rectangle describes a task or activity, while each diamond expresses a condition of the constructive procedure. Once the solution has been fully constructed, the evaluation of the solution is performed from scratch following the procedure described at the end of Section 1.1.

Algorithm 2 Greedy constructive procedure.

```
1: Procedure GreedyConstructive (G, H)
2: u ← random(V_G)
3: (i, j) = GetInitialHostGraphVertex(V_H)   ▷ Answer to question #1
4: φ(u) ← (i, j)
5: U_G ← V_G \ {u}
6: while U_G ≠ ∅ do
7:     u ← GetNextInputGraphVertex(φ, U_G)   ▷ Answer to question #2
8:     (i, j) ← GetNextHostGraphVertex(φ, u)   ▷ Answer to question #3
9:     φ(u) ← (i, j)
10:    U_G ← U_G \ {u}
11: end while
12: return φ
```

The choice of the first vertex of the input graph might influence deeply the quality of the generated solution. Moreover, our constructive procedure is not fully deterministic since random decisions are also made to break ties in both g_1 and g_2. Therefore, we propose to execute the constructive procedure for a fixed number of iterations, selecting as initial solution the best one among

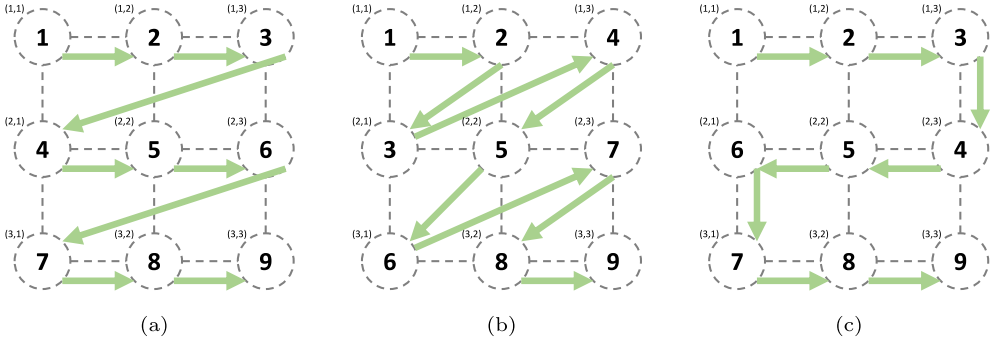

Fig. 2. Examples of the order followed to assign vertices of the input graph to vertices of the host graph using the Sequential (a), Diagonal (b) and Zigzag (c) patterns.

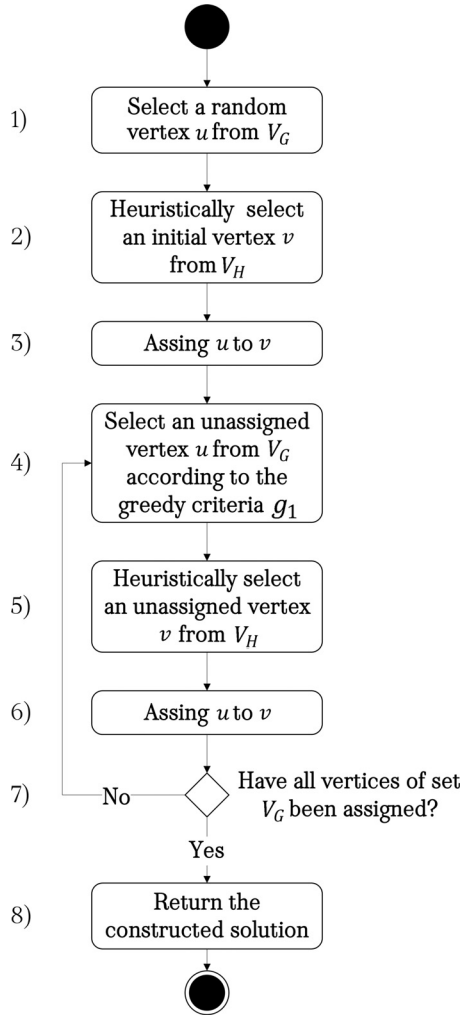

Fig. 3. Activity diagram of the constructive procedure..

them. It is worth mentioning that this strategy has been successfully used not only in IG (Huerta-Muñoz, Ríos-Mercado, & Ruiz, 2017; Stützle & Ruiz, 2018) but also in combination with VNS (López-Sánchez, Sánchez-Oro, & Hernández-Díaz, 2019; Pérez-Peló, Sánchez-Oro, Gonzalez-Pardo, & Duarte, 2021), or TS (Abdinnour-Helm & Hadley, 2000; Delmaire, Díaz, Fernández, & Ortega, 1999), among others. To ensure the construction of diverse solutions, the number of iterations is always set to a value larger than the number of vertices of the input graph, guaranteeing that at least one construction is performed starting from every vertex of the input graph.

2.2. Improvement strategy

Local search is a heuristic method widely used to solve hard optimization problems due to its ability to trade solution quality with computation time. This procedure is based on systematic moves from one solution to another with better quality, until reaching a locally optimal solution (Michiels, Aarts, & Korst, 2018). Given a predefined move operation, the set of feasible solutions reachable by the local search starting from that solution is usually known as neighborhood.

In the related literature, there have been proposed two different neighborhoods for the 2DBMP based on exchange and insert moves (see Rodríguez-García et al., 2021 for further details). However, the definition of these neighborhoods do not include the possibility of exploring the vertices of the host graph that have not been assigned to vertices of the input graph in the construction phase. In this paper, we propose a more flexible move operator, which explores those host vertices not initially assigned, resulting in a larger neighborhood space. Specifically, the number of vertices in the host graph ($|V_H| = \lceil\sqrt{n}\rceil \cdot \lceil\sqrt{n}\rceil$) is always larger than or equal to the number of vertices in the input graph ($|V_G| = n$). Therefore, φ is an injective function, since there may exist vertices in V_H that do not host a vertex of V_G. The move operator considers this property as follows: given a solution φ, a vertex $u \in V_G$ assigned to $(i, j) \in V_H$ (i.e., $\varphi(u) = (i, j)$), and a vertex $(i', j') \in V_H$ the move operator $Move(\varphi, u, (i', j'))$ assigns u to (i', j') (i.e., $\varphi(u) = (i', j')$). In the case that a vertex $v \in V_G$ was assigned to (i', j') prior the move, this move would also perform a new assignment for the vertex v (i.e., $\varphi(v) = (i, j)$). Otherwise, (i, j) will not host any vertex. This move produces a new feasible solution, which is denoted as $\varphi' \leftarrow Move(\varphi, u, (i', j'))$.

Let us illustrate this move with an example. Departing from the solution represented in Fig. 1c, we show in Fig. 4a the situation before performing $Move(\varphi, \mathrm{B}, (2, 3))$, where vertex B of the input graph is assigned to vertex (3,2) prior the move. Then, in Fig. 4b, we depict the resulting solution after the move, where B is assigned to (2,3) leaving (3,2) without any assignation.

Considering the aforementioned move operator, the proposed neighborhood for the 2DBMP is defined as follows:

$$N(\varphi) = \{Move(\varphi, u, (i', j')) : \forall\, u \in V_G, (i', j') \in V_H, \varphi(u) \neq (i', j')\}. \quad (9)$$

The size of $N(\varphi)$ can be determined depending on the number of vertices of the input graph. Specifically, given an input graph with $|V_G| = n$ vertices and the associated host graph with $|V_H| = \lceil\sqrt{n}\rceil \cdot \lceil\sqrt{n}\rceil$ vertices, the size of $N(\varphi)$ is $\frac{1}{2}n(\lceil\sqrt{n}\rceil^2 - 1)$.

In this research, we study two well-known strategies to explore the proposed neighborhood: *best improvement* and *first improvement* (Hansen & Mladenović, 2006). If a local search follows the *best improvement* strategy it selects, at each iteration, the best pos-

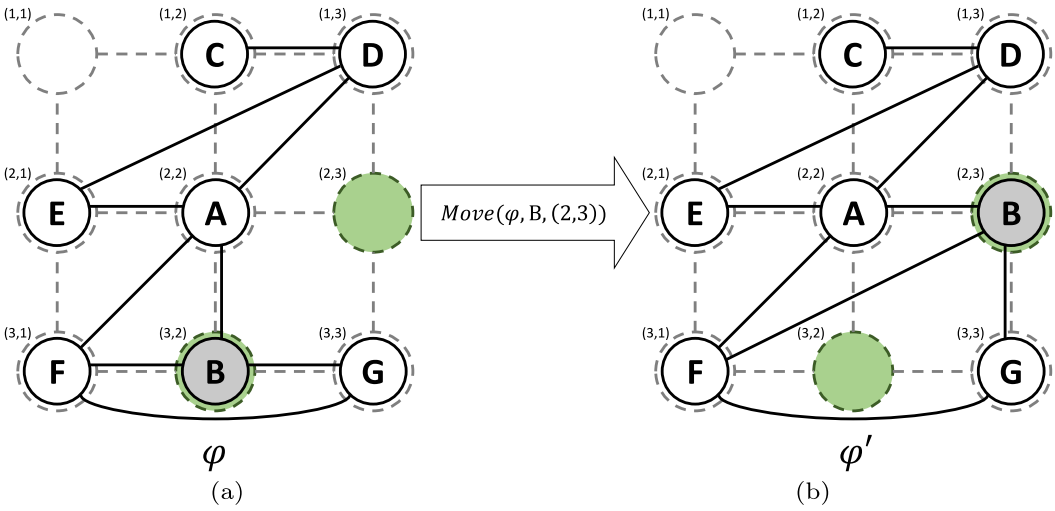

Fig. 4. (a) Example of an embedding φ. (b) The resultant embedding φ' obtained after the operation $Move(\varphi, \mathrm{B}, (2,3))$ (b).

sible move which produces an improvement of the current neighborhood; otherwise, in the *first improvement* strategy, it selects the first solution that improves the current solution. Any of them stop the search when no further improving moves can be performed. We study the effectiveness of both strategies in the experiments reported in Section 4.1.

2.3. Destruction and reconstruction procedures

The two main procedures of any IG algorithm are the destruction and reconstruction phases. The IG proposed in the context of 2DBMP, first removes a determined number of assignations of vertices of the input graph to vertices of the host graph, during the destruction phase. Then, the reconstruction phase applies a greedy heuristic to reassign the unassigned vertices until reaching a new feasible solution.

The number of assignations that should be removed is dynamically defined depending on the current number of iterations without improvement (see lines 6 and 7 of Algorithm 1). The rationale behind this decision is to have a trade-off between search intensification and diversification. In particular, when the number of iterations without improvement is large, it is expected that the procedure gets stuck in a "deep basin of attraction". Therefore, a large number of assignations are removed (with the corresponding reconstruction steps), with the aim of moving to rather distant solutions in the solution space. On the contrary, when the number of iterations without improvement is small, a few assignations are removed, which leads to a more localized search Stützle & Ruiz (2018). In addition, to avoid the complete destruction of the solution, we set a maximum number of assignations that can be removed. Specifically, this value is set to 25% of the vertices of the instance under consideration.

In this paper, we propose three different destruction strategies. The first one, denoted as *fully randomized destruction*, is a straightforward adaptation of the IG framework. Specifically, it consists of randomly selecting and then removing a determined number of assignations of vertices of the input graph to vertices of the host graph. The second strategy, denoted as *random area destruction*, focuses on a specific area of the host graph. More precisely, it selects a vertex of the host graph at random and then removes its assignation, and also the assignation of all adjacent host graph vertices (i.e., those at distance 1 to the selected initial vertex, according to Eq. (4)). This strategy keeps on removing host adjacent vertices to the unassigned area following a proximity criterion (i.e., first those at distance 2, then those at distance 3, etc.) until reaching the expected number of unassigned vertices. The third strategy, denoted as *greedy destruction*, focuses on those vertices that determine the value of the objective function. Notice that the 2DBMP consists of minimizing a maximum value; then, the objective function is usually determined by a reduced number of assignations. This destruction strategy removes the assignation of the vertex that determines the value of the objective function, also removing the assignations of all its adjacent host graph vertices. As in the second strategy, it keeps on removing vertices and their adjacent vertices in the host graph following a proximity criterion, until reaching the expected number of unassigned vertices.

As far as the reconstruction strategy is concerned, the Iterated Greedy framework Stützle & Ruiz (2018) suggests that the reconstruction should be governed by a greedy heuristic and, typically, deterministic (except random tiebreaking). We therefore do not explore any random reconstruction strategy. Specifically, given a partial solution obtained as the result of a destruction phase, the reconstruction of the solution is performed by following a greedy strategy. To this end, we propose the use of three strategies, based on the greedy criteria presented in Section 2.1: 1) unassigned vertices of the input graph are selected according to g_1 function and then are randomly assigned to any of the available host graph vertices; 2) unassigned vertices of the input graph are randomly selected and then assigned to its best host graph vertex according to g_2 function; and 3) the best vertex of the input graph is selected according to g_1 and it is assigned to the best host vertex selected according to g_2 function.

3. Advanced search strategies for exploring the neighborhoods

As it is well documented in the related literature, most of the computing time of a heuristic algorithm is spent by the local search procedure. Particularly, a local search heuristic selects, at each iteration, a move to a neighbor solution, if it results in an improvement of the objective function. Then, it needs to explore multiple neighbor solutions, evaluating each of them, to determine which move should be done next. In this section, we provide three new advanced strategies devoted to increasing the efficiency of the proposed local search: first, we introduce an efficient strategy to reduce the number of solutions to explore in a given neighborhood (see Section 3.1); second, once the number of moves has been reduced, we propose a technique to speed up the search by optimizing the calculation of the objective function after a move (see

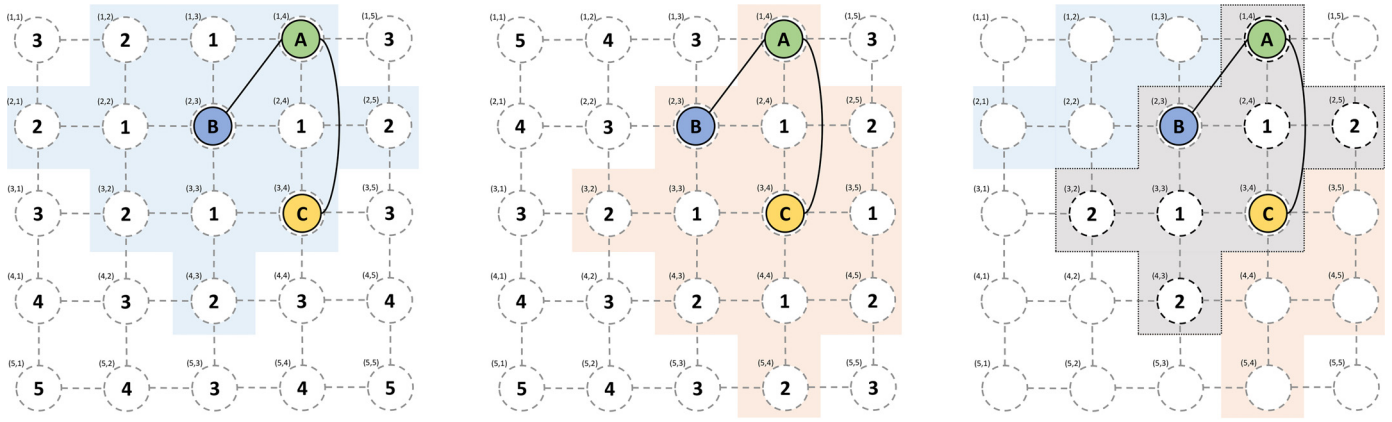

Fig. 5. Definition of the PSV of A for a solution φ, with $\mathrm{BW}(G, \varphi) = 2$.

Section 3.2); finally, when the move results in a tie, in terms of the objective function value, we propose a way of distinguishing between two solutions with the same quality (see Section 3.3). It is worth mentioning that some of these strategies can be applied or adapted with a few modifications, not only for other Graph Layout Problems, but also for other optimization problems.

3.1. Neighborhood reduction strategy

The neighborhood proposed in Section 2.2 for the 2DBMP has a size of $\frac{1}{2}n(\lceil\sqrt{n}\rceil^2 - 1)$, being n the number of vertices of the input graph. In the worst case, an exhaustive exploration requires to traverse the whole neighborhood. In this section, we propose a strategy to focus the search on promising solutions, avoiding to waste time in the evaluation of solutions which produce a deterioration in the objective function.

For each vertex of the input graph $u \in V_G$, we can define a set of vertices of the host graph, denoted as Promising Set of Vertices (PSV) which can host u satisfying that the distance d (in the host graph) from u to any of its neighbors in the input graph is equal or smaller to the objective function value.

Given a solution φ, and an input vertex $v \in V_G$, let us define S^{φ}_{v} as the set of host vertices which satisfy that the distance d from $\varphi(v)$ to any of them is smaller or equal to the $\mathrm{BW}(G, \varphi)$. More formally:

$$S^{\varphi}_{v} = \{(i, j) \in V_H : d(\varphi(v), (i, j)) \leq \mathrm{BW}(G, \varphi)\}. \tag{10}$$

Then, once we have defined the set S^{φ}_{v} for a single input vertex, we can define the set PSV for an input vertex $u \in V_G$ as the intersection of the sets S^{φ}_{v} for all adjacent vertices to u in the input graph (i.e., $v \in A(u)$). More formally:

$$\mathrm{PSV}(u, \varphi) = \bigcap_{v \in A(u)} S^{\varphi}_{v}. \tag{11}$$

In Fig. 5 we show an example of the definition of PSV of the vertex A. Particularly, vertices B and C are adjacent to A in the input graph. For the sake of clarity, the rest of the input vertices, not adjacent to A, have not been represented in the figure. Moreover, let us suppose that the value of the objective function for the solution φ, represented in the figure is 2 (i.e., $\mathrm{BW}(G, \varphi) = 2$).

In Fig. 5a we have highlighted with light blue color the set of host vertices placed at a distance 2 or minor from the host vertex (2,3), which hosts B. Specifically, $S^{\varphi}_{\mathrm{B}} = \{(1, 2), (1, 3), (1, 4), (2, 1), (2, 2), (2, 3), (2, 4), (2, 5), (3, 2), (3, 3), (3, 4), (4, 3)\}$. Additionally, in Fig. 5a we have indicated, with a number inside each host vertex, the distance d to (2,3). Similarly, in Fig. 5b we have highlighted in light orange the set of host vertices placed at a distance 2 or minor from the host vertex (3,4), which hosts C. Particularly, $S^{\varphi}_{\mathrm{C}} = \{(1, 4), (2, 3), (2, 4), (2, 5), (3, 2), (3, 3), (3, 4), (3, 5), (4, 3), (4, 4), (4, 5), (5, 4)\}$. Finally, in Fig. 5c we show the set PSV of A, obtained as the intersection of the two previous sets, which contains the host vertices placed at distance 2 or minor to any of the adjacent vertices to A. More formally, $\mathrm{PSV}(\mathrm{A}, \varphi) = S^{\varphi}_{\mathrm{B}} \cap S^{\varphi}_{\mathrm{C}} = \{(1, 4), (2, 3)(2, 4), (2, 5), (3, 2), (3, 3), (3, 4), (4, 3)\}$. Therefore, if A is assigned to any of the vertices in PSV(A), the maximum distance with respect to its adjacent vertices will be equal or smaller than the BW of φ.

Finally, we formally define $N_R(\varphi)$, as the restricted neighborhood of $N(\varphi)$ as follows:

$$N_R(\varphi) = \{Move(\varphi, u, (i, j)) : \forall\, u \in V_G, (i, j) \in \mathrm{PSV}(u), \varphi(u) \neq (i, j)\}. \tag{12}$$

Then, we propose the exploration of the reduced neighborhood instead of the whole neighborhood defined by the move operator introduced in Section 2.2.

3.2. Efficient move calculation

For the 2DBMP, a naive straightforward evaluation of a solution after a move consists in recalculating the value of the objective function from scratch. This means that the contribution of every input vertex has to be updated. However, an intelligent evaluation could avoid reevaluating the whole solution by just updating the elements that have been affected by the move. Therefore, we propose an efficient local search method, based on the move operator defined in Section 2.2, which applies an efficient evaluation after a move. Given an input graph G, a host graph H, and a solution φ, this move considers the vertex $u \in V_G$ (hosted in vertex $(i, j) \in V_H$), and assigns it to vertex $(i', j') \in V_H$. As it was aforementioned, if there were a vertex v hosted in (i', j'), i.e., $\varphi(v) = (i', j')$, this move would also assign v to (i, j). Therefore, an efficient objective function revaluation just needs to update the distance assigned to those edges with u (also v when existing) as an endpoint.

To define the Efficient Bandwidth calculation, denoted as EBW, of an input graph G and a solution φ' obtained by a move $Move(\varphi, u, (i, j))$, we first define the set of edges $I^{\varphi}_{u,(i,j)}$ involved in the move denoted as follows:

$$I^{\varphi}_{u,(i,j)} = \{(u, w) \in E_G : \forall w \in V_G\} \cup \{(v, w) \in E_G : \forall\, v, w \in V_G \wedge\ \varphi(v) = (i, j)\}. \tag{13}$$

Then, the EBW$(G, Move(\varphi, u, (i', j')), I)$ can be computed as:

$$\text{EBW}\ (G, Move(\varphi, u, (i', j')), I^{\varphi}_{u,(i,j)}) = \max\{\underbrace{\max_{(w,z)\in E_G \setminus I^{\varphi}_{u,(i,j)}} \{d(\varphi(w), \varphi(z))\}}_{\text{Not updated}}, \underbrace{\max_{(w,z)\in I^{\varphi}_{u,(i,j)}} \{d(\varphi(w), \varphi(z))\}}_{\text{Updated}}\}. \quad (14)$$

Notice that the distances of the edges that do not need to be updated in Eq. (14) can be easily evaluated by storing the distance associated with each edge before the move. Specifically, we use an array of sets, where the range of the array is determined by the possible distance values, while in each set it is stored all edges with a the same distance value. When we consider together the use of the aforementioned data structure and the efficient move calculation, the running time can be reduced in two orders of magnitude, on average, as we will show in the computational experience.

Let us illustrate this with an example. Particularly, we consider again the move $\varphi' \leftarrow Move(\varphi, \text{B}, (2, 3))$ depicted in Fig. 4. In order to evaluate the objective function of the resulting solution φ' it is just needed to update the distance associated to the edges with an endpoint in B, since (2,3) is not hosting any vertex of the input graph. More precisely, the edges with an endpoint in B are (A, B), (B, F), and (B, G). Then, only the distance of those edges needs to be re-evaluated, being the BW(G, φ') calculated as follows:

$$\begin{aligned}\text{BW}(G, \varphi') &= \max\{d(\varphi(\text{A}), \varphi(\text{D})), d(\varphi(\text{A}), \varphi(\text{E})), d(\varphi(\text{A}), \varphi(\text{F})),\\ &\quad d(\varphi(\text{C}), \varphi(\text{D})), d(\varphi(\text{D}), \varphi(\text{E})), d(\varphi(\text{F}), \varphi(\text{G})),\\ &\quad \boldsymbol{d(\varphi'(\text{A}), \varphi'(\text{B})), d(\varphi'(\text{B}), \varphi'(\text{F})), d(\varphi'(\text{B}), \varphi'(\text{G}))}\}\\ &= \max\{\underbrace{2, 1, 2, 1, 3, 2}_{\text{Not updated}}\underbrace{\mathbf{1, 3, 1}}_{\text{Updated}}\} = 3. \quad (15)\end{aligned}$$

Notice that, in this particular example, the number of edges evaluated is 3, while evaluating the whole solution requires 9 updates. Reasonably, the impact of this strategy is larger when dealing with instances composed by many vertices.

3.3. Tiebreak criterion for solutions with the same objective function value

An optimization problem consists of maximizing or minimizing a particular objective function. In some cases, this mathematical function consists of computing either the maximum or minimum value of a set of elements. Therefore, regardless the size of this set, the maximum or the minimum value, which determines the value of the objective function, is usually reached in more than one element. When the goal of a problem is to minimize a function based on a maximum value, we denote it as min-max problem. Similarly, when the goal of a problem is to maximize a minimum function, it is denoted as max-min problem. Max-min and min-max problems are quite common in optimization and become a challenge for heuristic methods because there may be many different solutions with the same objective function value, despite they are different solutions. This fact is usually known as "*flat landscape*" (Martí, Pantrigo, Duarte, & Pardo, 2013; Pardo et al., 2013). When this happens, it is difficult to determine the search direction since there is no way, according to the objective function, to determine which solution is more promising. To mitigate this problem, researchers have opted for tiebreaking criteria or alternative objective functions (Cavero, Pardo, & Duarte, 2021a; Cavero et al., 2021b).

The 2DBMP is a min-max optimization problem and preliminary experiments corroborate the existence of flat landscapes throughout the solution space. Furthermore, for an input graph with $n = |V_G|$ vertices and a host graph with $m = \lceil\sqrt{n}\rceil \cdot \lceil\sqrt{n}\rceil$ vertices, the number of solutions in the search space is upper bounded by $m!/(m-n)!$, but the range of values obtained as the evaluation with the objective function for that solutions are integer numbers in the interval $[1, 2 \cdot (\lceil\sqrt{n}\rceil - 1)]$. Let us remember that the evaluation of the objective function of the 2DBMP is directly related to the distance of adjacent input vertices measured in the host graph (see Eq. (4)). Then, the number of possible objective function values is considerably smaller than the number of solutions. Therefore, there might be many solutions with the same value of the objective function. To overcome this difficulty, we propose a tiebreak criterion based on the frequency of a particular distance in a solution. More formally, given an input graph $G(V_G, E_G)$ and a solution φ, we define f_l as the number of edges (u, v) of E_G with an associated distance in the host graph equal to l:

$$f_l = |\{(u, v) \in E_G : d(\varphi(u), \varphi(v)) = l\}|. \quad (16)$$

Let $l_{\max}$ be the maximum distance among all adjacent vertices in the graph. It trivially holds that $l_{\max}$ corresponds to the objective function value of the problem for a particular solution (i.e., $l_{\max} = \text{BW}(G, \varphi)$). Then, given an input graph G and an embedding φ, we propose a tiebreaking function t defined as follows:

$$t(G, \varphi) = \sum_{l=1}^{l_{max}} n^l \cdot f_l. \quad (17)$$

This equation is inspired by previous works related to circular layout problems (Cavero et al., 2021a; Cavero et al., 2021b). It takes into consideration not only the maximum distance of an embedding, but also additional semantic information related to promising solutions. Specifically, when the objective function value of two solutions is equal, both solutions are evaluated using the function t. The solution with the lower value of t is then chosen as the most promising one. The rationale behind this decision is to penalize those solutions with many edges with an associated distance close to the value of the objective function. It is worth mentioning that if the t value for two solutions is the same, they are considered as equivalent (in terms of the tiebreak criterion).

Let us illustrate the use of the tiebreak criterion with an example. To do so, we consider three different solutions φ_1, φ_2, and φ_3. Let us assume that the objective function for each solution is $\text{BW}(G, \varphi_1) = \max\{1, 3, 1, 2, 3\} = 3$, $\text{BW}(G, \varphi_2) = \max\{1, 2, 1, 2, 3\} = 3$, and $\text{BW}(G, \varphi_3) = \max\{2, 2, 1, 2, 3\} = 3$, respectively. Then, these three solutions are equal according to the objective function of the problem (i.e., the maximum value across all elements, which is 3). However, if we evaluate them with the tiebreak criterion, we appreciate differences among the solutions:

$$\begin{aligned}t(G, \varphi_1) &= 5^1 \cdot 2 + 5^2 \cdot 1 + 5^3 \cdot 2 = 10 + 25 + 250 = 285.\\ t(G, \varphi_2) &= 5^1 \cdot 2 + 5^2 \cdot 2 + 5^3 \cdot 1 = 10 + 50 + 125 = 185.\\ t(G, \varphi_3) &= 5^1 \cdot 1 + 5^2 \cdot 3 + 5^3 \cdot 1 = 5 + 75 + 125 = 205.\end{aligned}$$

In this case, since $t(G, \varphi_2) < t(G, \varphi_3) < t(G, \varphi_1)$, we would consider φ_2 as the most promising one. Similarly, we consider φ_3 more promising than φ_1.

4. Computational results

In this section, we present the experiments carried out to empirically evaluate the algorithmic proposals introduced in this paper. Particularly, we first propose a set of preliminary experiments to configure the best variant of our algorithm and to illustrate the influence of the advanced search strategies. Then, our best variant is compared with the best previous algorithm identified in the state of the art.

Table 1
Influence of the initial host vertex (answer to Question #1 in Section 2.1).

	Random	Corner	Center
Avg. OF	13.23	13.00	u8
CPU Time (s)	0.37	0.37	0.38
Dev. (%)	6.92	5.02	11.55
#Best	7	8	6

Table 2
Influence of the host vertex selected after the first assignation based on the weights w_1 and w_2 in g_1 (answer to Question #3 in Section 2.1).

	Diagonal	Sequential	ZigZag	g_2
Avg. OF	34.23	21.46	18.77	10.08
CPU Time (s)	4.68	4.51	4.48	5.79
Dev. (%)	283.19	138.11	89.66	6.15
#Best	0	0	1	12

The computational tests have been performed over 90 instances previously reported in the related literature on the 2DBMP (Rodríguez-García et al., 2021). These instances are grouped into two different subsets which include: 45 topologically diverse small graphs (with $|V_G| \in [5, 21]$ and $|E_G| \in [6, 190]$); and 45 representative graphs from the Harwell-Boeing collection (with $|V_G| \in [48, 960]$ and $|E_G| \in [78, 7442]$). To ease future comparisons, all instances have been made publicly available at https://www.heuristicas.es/.

All experiments have been performed on an AMD EPYC 7282 16-core virtual CPU with 16GB of RAM. The operating system used was Ubuntu 20.04.2 64 bit LTS, and all algorithms were implemented in Java 16.

4.1. Preliminary experiments

In this section, we identify the best configuration of the components of the Iterated Greedy procedure proposed in this paper. Also, we illustrate the merit of the proposed advanced search strategies. The preliminary experiments have been performed over a reduced set of instances consisting of 15% of the total considered instances (i.e., 13 graphs). We will refer to this subset of instances as the preliminary set.

For each of the experiments carried out, we report the following metrics: the average value of the objective function (Avg. OF), the total execution time in seconds (CPU Time (s)), the average deviation to the best solution found in the experiment (Dev. (%)) and the number of best solutions found in the experiment (#Best).

The first set of experiments performed is devoted to configure the best variant of the greedy constructive procedure described in Section 2.1, by trying to find the best answer to the questions raised in that section. Notice that the parameters are studied one by one, varying the values for the selected parameter and fixing the value for the rest of parameters.

Particularly, in Table 1 we analyse the proposed strategies to determine which is the most suitable host vertex to perform the first assignation (denoted as the answer to Question #1 in Section 2.1). The results reported in Table 1 correspond to a 100 of constructions where the input vertex has been chosen at random, g_1 is configured with $w_1 = 0.5$ and $w_2 = 0.5$ (i.e., both have the same weight), and g_2 is used as the criterion to determine the host vertices in the following assignations. With this configuration, starting the construction from a corner host vertex seems to be the best alternative, since the constructive procedure is able to reach the largest number of best solutions and the smallest deviation to the best solution.

In this case, we do not need to perform an experiment to select the most suitable input vertex to start the construction. This issue has been solved by starting the construction, at least, once from every input vertex.

Then, we analyse the influence of the parameters w_1 and w_2 in g_1 which determine the selection of the following input vertices further than the first assignation (denoted as the answer to Question #2 in Section 2.1). Let us remember, that w_1 and w_2 balance the influence of the adjacent assigned/unassigned vertices respectively, for every input vertex being evaluated with $g1$. Particularly, in Fig. 6 we depict the average performance of the constructive procedure for five different configurations of these two parameters, when the number of constructions increases from 1 to 2500. Notice that in this experiment the input/host vertices of the first assignation are selected following the best configuration found in the previous experiment. As we can observe in the figure, the combination $w_1 = 1.00, w_2 = 0.00$ is systematically the best configuration and therefore it will be selected for future experiments. Since the sum of w_1 and w_2 equals 1, the selected configuration indicates that, for this problem, the value of g_1 is fully determined by the already assigned adjacent vertices to the vertex being evaluated. Furthermore, the benefits obtained by performing multiple constructions do not improve significantly after 1500 constructions.

Finally, we analyze the influence of the strategies proposed to select the host vertex in every assignation but the first (denoted as the answer to Question #3 in Section 2.1). Particularly, in Table 2, we evaluate the four strategies proposed for this task. The reported results are obtained as the average of the best solutions found for each instance after 1500 constructions. Again, the input/host vertices of the first assignation are selected following the best configuration found with the criteria previously defined, and g_1 is configured with $w_1 = 1$ and $w_2 = 0$.

On the one hand, according to the selection of the vertices of the host graph, g_2 is easily recognized as the best strategy since it finds the best quality solutions (lower average of the objective function, lower deviation, and larger number of best solutions found). Among the pattern-based strategies, zigzag is the most prominent.

To sum up, the final configuration of our constructive procedure has been set to be executed for 1500 constructions, and the best overall solution is selected. Each construction starts from a different initial input vertex (if all vertices have been used at least once, the procedure selects a repeated vertex to start with). The initial host vertex is set to be one of the corner vertices of the grid. The following input vertices are selected one by one with g_1 configured with $w_1 = 1$ and $w_2 = 0$. Finally, g_2 is selected as the method to determine the host vertices for any assignation performed after the first one.

Our next preliminary experiment is devoted to test the influence of the advanced strategies proposed in Section 3 in the local search procedure described in Section 2.2. First, we evaluated the exploration of the neighborhood defined by the move operator following both: a *first improvement* and a *best improvement* strategy. We found that both strategies reached the same average quality of the objective function (32.54) for the preliminary data set. However, the CPU time of the local search using a best improvement strategy was 5 times larger than using a *first improvement* strategy. Then, we configured our local search procedure with a *first improvement* strategy.

In Table 3, we report the results obtained when incorporating each of the three proposed advanced strategies to the local search procedure (i.e., the tiebreak criterion (T), the efficient move calculation (E), and the neighborhood reduction strategy (R)). We also include in the comparison the original local search procedure in isolation (LS). The results provided in the table are obtained

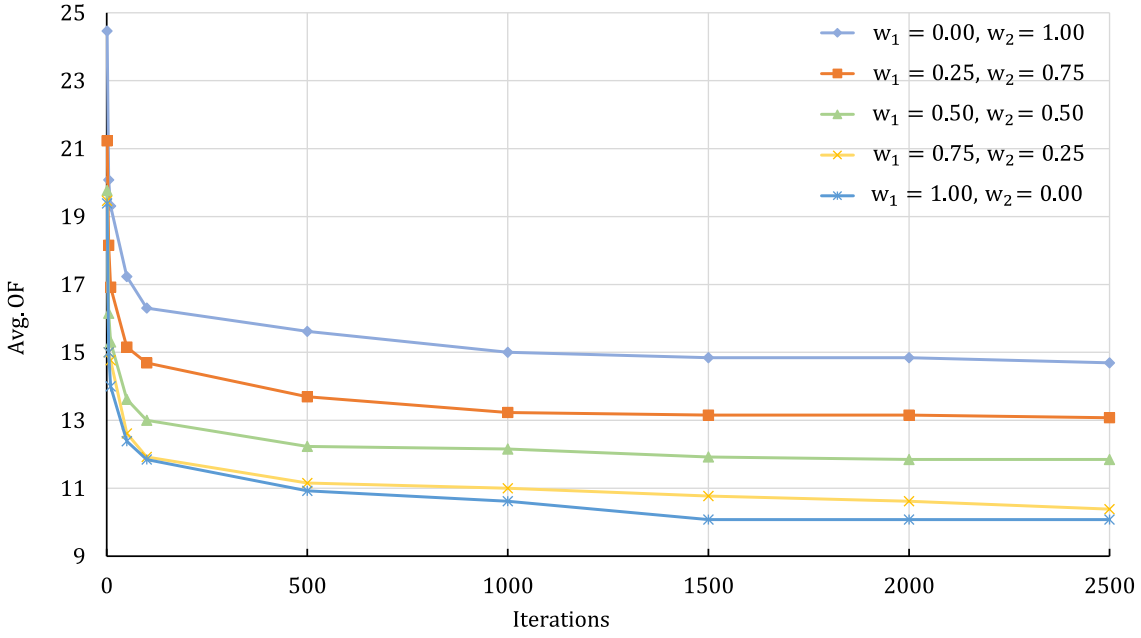

Fig. 6. Evolution of the average objective function value when increasing the number of constructions for different values of w_1 and w_2 in g_1.

Table 3
Contribution of advanced strategies to the local search.

	LS	LS+T	LS+T+E	LS+T+E+R
Avg. OF	32.54	7.77	7.77	7.85
CPU Time (s)	72.51	7873.28	22.76	2.18
Dev. (%)	350.04	4.62	4.62	7.05
#Best	0	8	8	9

after a single execution of each method, where the initial solution was the same for all compared methods and it was randomly constructed.

As we can observe in Table 3 the inclusion of the tiebreak criterion (LS+T) drastically improves the quality of the solutions obtained with respect to the original local search (LS) although the time increases considerably. This increase in the time needed by the method is due to the larger exploration of solutions performed by the LS+T. As expected, LS+T and LS+T+E were able to reach the same solutions in terms of quality, however, then the efficient move calculation reduces the time needed to reach them in 99.71%. Finally, the method including the proposed neighborhood reduction strategy, LS+T+E+R, is able to reduce by one order of magnitude the time needed by LS+T+E, slightly deteriorating the average quality of the solutions obtained. Both behaviors are explained by the fact that the number of solutions explored is considerably smaller. We consider that LS+T+E+R is the most promising combination as a balance between computing time and quality.

Next, we study the best procedures for the destruction and reconstruction phase. In this experiment, we analyse all possible combinations of the strategies proposed in Section 2.3. Specifically, we evaluate three destruction strategies: random assignations (random), assignations of random areas (random area), and assignations of areas contributing to the objective function of the solution (greedy area). Additionally, as far as the reconstruction phase is concerned, we evaluate three proposals: g_1 + random, random + g_2, and $g_1 + g_2$. Each proposal includes two strategies to determine the next assignation. The first strategy selects an input unassigned vertex, while the second strategy selects an available host vertex.

In Table 4, we present the results of this experiment. Particularly, each Iterated Greedy configuration has been executed for a maximum of 300 iterations, with the additional condition that the method is halted if it does not find an improvement of the best solution found in the last 150 iterations. The best configuration is obtained when the destruction is made greedily ("Greedy area" in the table) and the reconstruction is made by using the "random + g_2" criterion. The second best variant is the one where the destruction is made at random ("Random" in the table) and the reconstruction uses "random + g_2". However, this variant finds very similar solutions in terms of quality in half time. Therefore, as a trade-off between quality and time, we have selected this second configuration for our final proposed procedure.

Finally, it is important to remark that we performed a fine-tuning experiment to adjust the parameters: maximum number of iterations (*maxIter*) and the maximum number of iterations without improving (*maxNotImprIter*) introduced in the Algorithm 1. Particularly, we tested different values of *maxIter* in the range [100, 1000] in steps of 50. Similarly, we studied the behavior of *maxNotImprIter*) with different percentages (0.25, 0.5, and 0.75) of the *maxIter*. For the sake of brevity, we do not include all the values of this experiment in here. However, among the proposed configurations, we selected $maxIter = 300$ and $maxNotImprIter = 0.75 \cdot maxIter = 225$ for our final design, as a balance of quality and CPU time.

To conclude the preliminary experiments, we compare our three main algorithmic proposals to verify if an increase in the complexity of the method also results in an improvement in the obtained results. Specifically, we propose two executing scenarios: a single run of each method and running each method iteratively for 100 seconds. Notice that in this experiment, the solution produced by the greedy constructive is provided to the local search and to the Iterated Greedy procedure. The results obtained are reported in Table 5. As expected, in the single execution scenario, the IG is the best method in terms of average of the objective function, deviation and # Best solutions found. However, it is also the most time-consuming procedure. On the other hand, when running all methods for 100 seconds, the differences among the results obtained are reduced, but IG is still the best overall method.

Table 4
Influence of the destruction and reconstruction strategies in the performance of the greedy constructive procedure.

Destruction	Reconstruction	Avg. OF	CPU Time (s)	Dev. (%)	#Best
Random	g_1 + random	5.46	154.21	9.29	8
	random + g_2	5.23	60.20	3.85	11
	$g_1 + g_2$	5.46	43.13	7.37	8
Random area	g_1 + random	5.38	145.25	6.41	9
	random + g_2	5.31	110.44	4.81	10
	$g_1 + g_2$	5.46	65.79	7.37	8
Greeedy area	g_1 + random	5.31	147.90	4.81	10
	random + g_2	5.15	119.62	0.96	12
	$g_1 + g_2$	5.38	63.62	6.09	9

Table 5
Behaviour of the proposed strategies in a single execution and running for 100 seconds.

	Single execution			100 seconds		
	Constructive	LS+T+E+R	IG	Constructive	LS+T+E+R	IG
Avg. OF	10.08	6.00	5.23	9.08	5.38	5.23
CPU Time (s)	6.05	6.43	84.71	102.79	106.86	106.90
Dev. (%)	95.69	13.32	0.00	75.62	4.49	1.92
#Best	0	5	13	0	10	12

Table 6
Results obtained by the compared methods on the 41 instances of the Small graphs data set with a known optimal, and on the 45 instances of the Harwell-Boeing data set.

	Small graphs (41)			Harwell-Boeing (45)	
	M3 (Rodríguez-García et al., 2021)	BVNS (Rodríguez-García et al., 2021)	IG	BVNS (Rodríguez-García et al., 2021)	IG
Avg. OF	2.17	2.32	2.17	7.11	4.84
CPU Time (s)	1957.65	3.64	0.02	439.63	102.01
Dev. (%)	0.00	12.20	0.00	51.77	0.00
#Best	41	35	41	5	45

4.2. Final experiments

In this section, we compare our best Iterated Greedy (IG) variant with the best previous algorithms in the state of the art: the best Constraint Satisfaction Programming (CSP) model proposed in Rodriguez-Tello et al. (2019) (denoted M3) and the Basic Variable Neighborhood Search (BVNS) proposed in Rodríguez-García et al. (2021). Both procedures were described in the literature review. Notice that we have compared our procedure with the original source code implemented and provided by the authors. To make the fairest comparison possible, instead of directly using the results reported in Rodríguez-García et al. (2021), the BVNS procedure was run again with the configuration indicated by the authors, in the same execution environment as the one used for our code.

In Table 6 we report the quality indicators presented in the preliminary experiment: the average deviation, the average execution time, and the number of best solutions found for each of the subsets of instances studied: the diverse small graph subset and the Harwell-Boeing subset.

In particular, on the left side of Table 6, we compare our IG with the BVNS and M3 over the set of diverse small graphs. Note that M3 was unable to complete the search for 4 instances out of 45 within the established time limit (72h). Therefore, we have removed those instances from this comparison to fairly illustrate the behavior of the proposed algorithm. Additionally, in Table A.1 we include the individual results per instance for each of the 45 instances of the complete subset. We observe in Table 6 that M3 and IG were able to reach the optimal solution for the 41 instances studied, followed by BVNS with 35. However, the time required by IG was 5 orders of magnitude shorter than M3 and 2 orders of magnitude shorter than BVNS.

Similarly, on the right side of Table 6, we compare IG and BVNS over the Harwell-Boeing subset. In this case, M3 was not able to finish within the maximum time limit and therefore it has been excluded from this comparison. Again, to ease future comparisons, we include the individual of each instance in Table A.2. Based on the results reported in Table 6 we observe that IG finds the best solution for all the graphs studied (45) in less computational time (102.01 s) than the BVNS procedure (439.63 s). Consequently, the average value of the objective function is lower in the solutions obtained by IG than in the solutions obtained by BVNS. Finally, we highlight that BVNS has a 51.77% deviation from the best solutions found, obtaining only five best solutions out of 45 instances.

To complement the previous experiment, we conducted a Wilcoxon signed rank test. The resulting p-value < 0.00001 confirms the significance of the results obtained when comparing the methods for the tested instances.

5. Conclusions

In this paper, we tackle the Two-Dimensional Bandwidth Minimization Problem by proposing several efficient heuristic strategies to find high-quality solutions for the problem. The 2DBMP belongs to the graph layout family of problems, and it has been previously approached from an exact perspective, based on Constraint Satisfaction Programming, and from a heuristic perspective, based on the Variable Neighborhood Search metaheuristic.

We have developed an efficient and effective Iterated Greedy algorithm to deal with the 2DBMP, including an exhaustive study of multiple greedy criteria at the destruction and reconstruction steps within the IG framework. In addition, we introduce a novel local search procedure based on swap moves of vertices, which includes three advanced enhancement strategies. It is worth mentioning that several of the strategies proposed in this paper have further

applicability to other optimization problems, especially those related to Graph Layout Problems.

The results obtained in this paper emphasize the importance of using a tiebreak criterion to guide the search through flat landscape regions. This is a key strategy when the objective function is not useful in distinguishing between two solutions with the same objective function value. Also, we identified that classical move operators applied to Graph Layout Problems, such as the 2DBMP, usually drive to extensive neighborhoods. In this kind of scenario, local search procedures might be inefficient when the time limit is short. To overcome this drawback, we propose two general strategies with applicability to other problems: a speed-up technique to evaluate the objective function of neighbor solutions; and a neighborhood reduction technique based on the exploration of the most promising neighbor solutions. Moreover, the graph structure of either the input and host graphs is key in determining the best heuristic strategy in the context of GLPs. Particularly, the number of adjacent vertices of each vertex tends to contribute to relevant information at the time of constructing new solutions.

Finally, we would like to highlight that the best algorithmic variant of our proposal has been compared to the best previous method in the state of the art, over a previously reported set of instances. The obtained results, supported by statistical tests, corroborate the merit of our proposal and establish it as a new state-of-the-art algorithm for the 2DBMP.

Acknowledgment

This research has been partially supported by the Ministerio de Ciencia, Innovación y Universidades (Grant Ref. PGC2018-095322-B-C22, PID2021-125709OA-C22 and FPU19/04098) and by the Comunidad de Madrid and the European Regional Development Fund (Grant Ref. P2018/TCS-4566). We also thank M.A. Rodrguez et al., authors of the previous most competitive method in the state of the art Rodríguez-García et al. (2021) for sharing their code with us.

Appendix A. Individual results per instance

In Table A.1 and Table A.2 we report the individual results per instance for the Small and Harwell-Boeing data sets. These values were used to calculate the values presented in Table 6.

Table A.1
Individual results per instance obtained from the small data set. Note that a symbol "-" in the table indicates that the algorithm was not able to solve the instance in a maximum CPU time of 72h.

		M3			BVNS			IG		
Instance	Best	OF	CPU Time (s)	Dev. (%)	OF	CPU Time (s)	Dev. (%)	OF	CPU Time (s)	Dev. (%)
p2p3	**1**	1	0.20	0.00	2	0.11	1.00	1	0.01	0.00
p3p3	**1**	1	0.26	0.00	2	0.38	1.00	1	0.01	0.00
p4p5	**1**	1	0.22	0.00	2	7.84	1.00	1	0.03	0.00
p2c3	**2**	2	0.24	0.00	2	0.16	0.00	2	0.01	0.00
p3c3	**2**	2	0.20	0.00	2	0.54	0.00	2	0.01	0.00
p4c5	**2**	2	0.53	0.00	3	7.55	0.50	2	0.03	0.00
c3c3	**2**	2	0.24	0.00	2	0.83	0.00	2	0.02	0.00
c3c4	**2**	2	0.28	0.00	2	2.13	0.00	2	0.03	0.00
c4c5	**2**	2	0.28	0.00	3	9.26	0.50	2	0.04	0.00
k3k4	**3**	3	0.55	0.00	3	2.59	0.00	3	0.02	0.00
k4k5	**4**	4	74192.71	0.00	4	20.00	0.00	4	0.05	0.00
c3k4	**3**	3	0.60	0.00	3	2.83	0.00	3	0.03	0.00
c4k5	**3**	3	2.18	0.00	3	17.86	0.00	3	0.05	0.00
p3k4	**2**	2	0.27	0.00	2	2.58	0.00	2	0.02	0.00
p4k5	**3**	3	1.93	0.00	3	14.81	0.00	3	0.04	0.00
path10	**1**	1	0.30	0.00	1	0.44	0.00	1	0.01	0.00
path15	**1**	1	0.35	0.00	2	1.32	1.00	1	0.02	0.00
path20	**1**	1	0.31	0.00	1	4.04	0.00	1	0.02	0.00
cycle10	**1**	1	0.24	0.00	1	0.40	0.00	1	0.01	0.00
cycle15	**2**	2	0.50	0.00	2	1.19	0.00	2	0.02	0.00
cycle20	**1**	1	0.28	0.00	1	4.30	0.00	1	0.03	0.00
wheel5	**2**	2	0.21	0.00	2	0.12	0.00	2	0.00	0.00
wheel7	**2**	2	0.23	0.00	2	0.35	0.00	2	0.01	0.00
wheel10	**2**	2	0.41	0.00	2	1.17	0.00	2	0.02	0.00
wheel15	**3**	3	4444.86	0.00	3	4.11	0.00	3	0.02	0.00
wheel20	3	-	-	-	3	11.93	0.00	4	0.03	0.33
cyclePow10-2	**2**	2	0.25	0.00	2	1.12	0.00	2	0.02	0.00
cyclePow15-2	**2**	2	0.28	0.00	2	3.58	0.00	2	0.02	0.00
cyclePow20-2	**2**	2	0.30	0.00	2	9.15	0.00	2	0.04	0.00
cyclePow10-10	**4**	4	0.21	0.00	4	3.74	0.00	4	0.02	0.00
cyclePow15-10	6	-	-	-	6	15.01	0.00	6	0.04	0.00
cyclePow20-10	6	-	-	-	6	20.00	0.00	6	0.08	0.00
bipartite3-3	**2**	2	0.26	0.00	2	0.18	0.00	2	0.01	0.00
bipartite3-4	**3**	3	0.33	0.00	3	0.28	0.00	3	0.02	0.00
bipartite4-4	**3**	3	0.33	0.00	3	0.56	0.00	3	0.02	0.00
bipartite5-5	**3**	3	0.78	0.00	3	1.46	0.00	3	0.02	0.00
bipartite7-8	**4**	4	1050.31	0.00	4	9.03	0.00	4	0.04	0.00
bipartite10-10	5	-	-	-	5	20.00	0.00	5	0.21	0.00
petersen	**2**	2	0.31	0.00	2	0.77	0.00	2	0.01	0.00
complete5	**2**	2	0.28	0.00	2	0.12	0.00	2	0.01	0.00
complete10	**4**	4	14.03	0.00	4	4.15	0.00	4	0.02	0.00
tree2-2	**1**	1	0.31	0.00	1	0.11	0.00	1	0.01	0.00
tree2-3	**2**	2	0.31	0.00	2	0.99	0.00	2	0.01	0.00
tree3-2	**2**	2	0.25	0.00	2	1.25	0.00	2	0.02	0.00
tree2-4	**2**	2	546.79	0.00	2	5.93	0.00	2	0.04	0.00

Table A.2
Individual results per instance obtained from the Harwell-Boeing data set.

		BVNS (Rodríguez-García et al., 2021)			IG		
Instance	Best	OF	CPU Time (s)	Dev. (%)	OF	CPU Time (s)	Dev. (%)
bcsstk01	5	5	48.00	0.00	5	0.35	0.00
can___62	2	3	62.00	50.00	2	0.22	0.00
nos4	4	4	100.01	0.00	4	1.13	0.00
bcspwr03	3	4	118.01	33.33	3	0.97	0.00
bcsstk04	8	9	132.01	12.50	8	33.24	0.00
bcsstk22	3	4	138.01	33.33	3	1.76	0.00
can__144	4	5	144.01	25.00	4	2.84	0.00
bcsstk05	7	7	153.01	0.00	7	13.82	0.00
can__161	4	6	161.01	50.00	4	4.31	0.00
dwt__198	4	5	198.01	25.00	4	6.49	0.00
dwt__209	5	6	209.01	20.00	5	12.95	0.00
dwt__221	4	5	221.01	25.00	4	7.26	0.00
can__229	5	7	229.01	40.00	5	11.65	0.00
dwt__234	4	4	234.01	0.00	4	11.46	0.00
nos1	3	4	237.01	33.33	3	4.68	0.00
dwt__245	4	6	245.02	50.00	4	8.69	0.00
lshp_265	3	6	265.00	100.00	3	9.31	0.00
bcspwr04	4	7	274.02	75.00	4	13.05	0.00
ash292	4	6	292.00	50.00	4	9.84	0.00
can__292	6	7	292.00	16.67	6	31.82	0.00
dwt__307	5	7	307.00	40.00	5	25.23	0.00
dwt__310	4	5	310.00	25.00	4	10.78	0.00
dwt__361	5	8	361.01	60.00	5	25.45	0.00
plat362	7	8	362.01	14.29	7	110.03	0.00
bcsstk07	6	9	420.01	50.00	6	298.42	0.00
bcspwr05	5	5	443.01	0.00	5	29.52	0.00
can__445	7	9	445.01	28.57	7	59.74	0.00
bcsstk20	4	6	485.01	50.00	4	39.75	0.00
494_bus	5	6	494.01	20.00	5	41.65	0.00
dwt__503	6	8	503.01	33.33	6	83.16	0.00
lshp_577	5	8	577.00	60.00	5	63.92	0.00
dwt__607	5	9	607.00	80.00	5	107.66	0.00
662_bus	5	7	662.01	40.00	5	56.83	0.00
nos6	5	14	960.01	180.00	5	49.94	0.00
685_bus	5	8	685.01	60.00	5	51.32	0.00
can__715	8	11	715.01	37.50	8	698.93	0.00
nos7	6	10	729.01	66.67	6	174.52	0.00
dwt__758	5	7	758.01	40.00	5	127.15	0.00
lshp_778	4	9	778.01	125.00	4	142.28	0.00
bcsstk19	6	9	817.00	50.00	6	399.65	0.00
dwt__878	5	9	878.00	80.00	5	192.83	0.00
gr_30_30	2	9	900.01	350.00	2	45.41	0.00
dwt__918	6	9	918.01	50.00	6	431.28	0.00
nos2	4	6	957.01	50.00	4	106.14	0.00
nos3	7	14	960.01	100.00	7	1032.95	0.00

References

Abdinnour-Helm, S., & Hadley, S. W. (2000). Tabu search based heuristics for multi--floor facility layout. *International Journal of Production Research, 38*(2), 365–383.

Bezrukov, S. L., Chavez, J. D., Harper, L. H., Röttger, M., & Schroeder, U. P. (1998). Embedding of hypercubes into grids. In L. Brim, J. Gruska, & J. Zlatuka (Eds.), *Mathematical foundations of computer science 1998*. In *Lecture notes in computer science* (pp. 693–701). Berlin, Heidelberg: Springer.

Bhatt, S. N., & Thomson Leighton, F. (1984). A framework for solving VLSI graph layout problems. *Journal of Computer and System Sciences, 28*(2), 300–343.

Booch, G. (2005). *The unified modeling language user guide*. Pearson Education India.

Cavero, S., Pardo, E. G., & Duarte, A. (2021a). Influence of the alternative objective functions in the optimization of the cyclic cutwidth minimization problem. In *Advances in artificial intelligence*. In *Lecture notes in computer science* (pp. 139–149). Cham: Springer International Publishing.

Cavero, S., Pardo, E. G., Laguna, M., & Duarte, A. (2021b). Multistart search for the cyclic cutwidth minimization problem. *Computers & Operations Research, 126*, 105116.

Cavero, S., Pardo, E. G., & Duarte, A. (2022a). A general variable neighborhood search for the cyclic antibandwidth problem. *Computational Optimization and Applications, 81*(2), 657–687.

Cavero, S., Pardo, E. G., Duarte, A., & Rodriguez-Tello, E. (2022b). A variable neighborhood search approach for cyclic bandwidth sum problem. *Knowledge-Based Systems, 246*, 108680.

Chung, F. (1988). Labelings of graphs. *Selected topics in graph theory, 3*, 151–168.

Craw, S. (2010). *Manhattan distance* (pp. 639–639)). Boston, MA: Springer US.

Del Corso, G. M., & Manzini, G. (1999). Finding exact solutions to the bandwidth minimization problem. *Computing, 62*(3), 189–203.

Delmaire, H., Díaz, J. A., Fernández, E., & Ortega, M. (1999). Reactive grasp and tabu search based heuristics for the single source capacitated plant location problem. *INFOR: Information Systems and Operational Research, 37*(3), 194–225.

Díaz, J., Petit, J., & Serna, M. (2002). A survey of graph layout problems. *ACM Computing Surveys (CSUR), 34*(3), 313–356.

Feo, T. A., & Resende, M. G. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization, 6*(2), 109–133.

Gurari, E. M., & Sudborough, I. H. (1984). Improved dynamic programming algorithms for bandwidth minimization and the mincut linear arrangement problem. *Journal of Algorithms, 5*(4), 531–546.

Hansen, P., & Mladenović, N. (2006). First vs. best improvement: An empirical study. *Discrete Applied Mathematics, 154*(5), 802–817.

Hansen, P., Mladenović, N., Todosijević, R., & Hanafi, S. (2017). Variable neighborhood search: Basics and variants. *EURO Journal on Computational Optimization, 5*(3), 423–454.

Hromkovič, J., Müller, V., Sỳkora, O., & Vrt'o, I. (1992). On embedding interconnection networks into rings of processors. In *International conference on parallel architectures and languages Europe* (pp. 51–62). Springer.

Huerta-Muñoz, D. L., Ríos-Mercado, R. Z., & Ruiz, R. (2017). An iterated greedy heuristic for a market segmentation problem with multiple attributes. *European Journal of Operational Research, 261*(1), 75–87.

Jinjiang, Y., & Sanming, Z. (1995). Optimal labelling of unit interval graphs. *Applied Mathematics, 10*(3), 337–344.

Lai, Y.-L., & Williams, K. (1999). A survey of solved problems and applications on bandwidth, edgesum, and profile of graphs. *Journal of Graph Theory, 31*(2), 75–94.

Lin, L., & Lin, Y. (2010). Two models of two-dimensional bandwidth problems. *Information Processing Letters, 110*(11), 469–473.

Lin, L., & Lin, Y. (2011). Square-root rule of two-dimensional bandwidth problem. *RAIRO-Theoretical Informatics and Applications, 45*(4), 399–411.
Lin, Y. (1994). The cyclic bandwidth problem. *Journal of Systems Science and Complexity, 7*(3), 282–288**.** Cited By 12
Lourenço, H. R., Martin, O. C., & Stützle, T. (2003). Iterated local search. In *Handbook of metaheuristics* (pp. 320–353). Springer.
López-Sánchez, A., Sánchez-Oro, J., & Hernández-Díaz, A. (2019). GRASP and VNS for solving the p-next center problem. *Computers & Operations Research, 104*, 295–303.
Martí, R., Pantrigo, J. J., Duarte, A., & Pardo, E. G. (2013). Branch and bound for the cutwidth minimization problem. *Computers & Operations Research, 40*(1), 137–149.
McAllister, A. et al. (1999). A new heuristic algorithm for the linear arrangement problem.
Michiels, W., Aarts, E. H., & Korst, J. (2018). Theory of local search. In *Handbook of heuristics* (pp. 299–339). Springer.
Mladenovic, N., Urosevic, D., Pérez-Brito, D., & García-González, C. G. (2010). Variable neighbourhood search for bandwidth reduction. *European Journal of Operational Research, 200*(1), 14–27.
Papadimitriou, C. H. (1976). The np-completeness of the bandwidth minimization problem. *Computing, 16*(3), 263–270.
Pardo, E. G., García-Sánchez, A., Sevaux, M., & Duarte, A. (2020). Basic variable neighborhood search for the minimum sitting arrangement problem. *Journal of Heuristics, 26*(2), 249–268.
Pardo, E. G., Martí, R., & Duarte, A. (2016). Linear layout problems. In R. Martí, P. Panos, & M. G. Resende (Eds.), *Handbook of heuristics* (pp. 1–25). Cham: Springer International Publishing.
Pardo, E. G., Mladenović, N., Pantrigo, J. J., & Duarte, A. (2013). Variable formulation search for the cutwidth minimization problem. *Applied Soft Computing, 13*(5), 2242–2252.
Pérez-Peló, S., Sánchez-Oro, J., Gonzalez-Pardo, A., & Duarte, A. (2021). A fast variable neighborhood search approach for multi-objective community detection. *Applied Soft Computing, 112*, 107838.
Petit, J. (2004). Experiments on the minimum linear arrangement problem. *ACM Journal of Experimental Algorithmics, 8*, 2.3.
Ren, J., Hao, J.-K., & Rodriguez-Tello, E. (2019). An iterated three-phase search approach for solving the cyclic bandwidth problem. *IEEE Access, 7*, 98436–98452.
Ren, J., Hao, J.-K., Rodriguez-Tello, E., Li, L., & He, K. (2020). A new iterated local search algorithm for the cyclic bandwidth problem. *Knowledge-Based Systems, 203*, 106136.
Rodriguez-Tello, E., Hao, J.-K., & Torres-Jimenez, J. (2008a). An effective two-stage simulated annealing algorithm for the minimum linear arrangement problem. *Computers & Operations Research, 35*(10), 3331–3346.
Rodriguez-Tello, E., Hao, J.-K., & Torres-Jimenez, J. (2008b). An improved simulated annealing algorithm for bandwidth minimization. *European Journal of Operational Research, 185*(3), 1319–1335.
Rodriguez-Tello, E., Narvaez-Teran, V., & Lardeux, F. (2019). Dynamic multi-armed bandit algorithm for the cyclic bandwidth sum problem. *IEEE Access, 7*, 40258–40270.
Rodríguez-García, M. A., Sánchez-Oro, J., Rodriguez-Tello, E., Monfroy, E., & Duarte, A. (2021). Two-dimensional bandwidth minimization problem: Exact and heuristic approaches. *Knowledge-Based Systems, 214*, 106651.
Ruiz, R., & Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research, 177*(3), 2033–2049.
Stützle, T., & Ruiz, R. (2018). *Iterated greedy* (pp. 547–577)). Cham: Springer International Publishing.
Tsang, E. (2014). *Foundations of constraint satisfaction: The classic text*. Books on Demand.